Universidad Nacional de Mar del Plata
Facultad de Ingeniería

PROYECTO FINAL

"SUBÍ QUE TE LLEVO"

GRUPO Nº 12 - SOFÍA ISABELLA PALLADINO

Programación III
Ciclo 2024

Patrones Utilizados en el trabajo

- Patrón Singleton
- Patrón Template
- Patrón Decorator
- Patrón MVC
- Patrón Observer/Observable

Patrón Singleton

Utilizado en la clase Empresa

- Se asegura que exista una unica instancia de esta clase a la vez que se proporciona acceso global a dicha clase.
- Garantiza que todos los componentes usen los mismos datos.

Patrón Template

Utilizado en la clase “Vehiculo”

- Aplicado para establecer la prioridad a la hora de buscar un vehiculo que sea apto para el pedido.
- Define los metodos a seguir para determinar si un vehiculo es apto para un pedido.

Patrón Decorator

Utilizado para la personalizacion de los Viajes dependiendo el pedido

Clases que lo implementan:

- DecoratorViajes (Padre)

Heredan:

- DecoratorEquipajeBaul
- DecoratorConMascota

Patrón Decorator

- Se encarga de añadir comportamiento adicional a un objeto de manera dinámica sin modificar su estructura original.

```java
public abstract class DecoratorViajes implements IViaje, Serializable {

    protected IViaje encapsulado;

    public DecoratorViajes(IViaje encapsulado) {
        this.encapsulado = encapsulado;
    }
```

```java
public class DecoratorEquipajeBaul extends DecoratorViajes {

    public DecoratorEquipajeBaul(IViaje encapsulado) {
        super(encapsulado);
    }


    @Override
    public double getCosto() {
        double costoEncapsulado=this.encapsulado.getCosto();
        double incrXPersona=Viaje.getCostoBase()*0.10*this.encapsulado.getCantidadPersonas();
        double incrXKm=Viaje.getCostoBase()*0.05*this.encapsulado.getDistanciaReal();

        return costoEncapsulado+incrXPersona+incrXKm;
    }

}
```

Patrón Factory

Utilizado para la creación de instancias de Usuarios. Viajes y Vehiculos

```java
public class ViajeFactory {

    public IViaje getViaje(Pedido pedido) {
        IViaje viaje;
        viaje=null;
        if(pedido.getZona().equalsIgnoreCase( anotherString: "Zona Estandar")) {
            viaje=new ViajeZonaEstandar(pedido);
        }
        else if(pedido.getZona().equalsIgnoreCase( anotherString: "Calle sin asfaltar")) {
            viaje=new ViajeZonaCalleSinAfaltar(pedido);
        }
        else if(pedido.getZona().equalsIgnoreCase( anotherString: "Zona Peligrosa")){
            viaje=new ViajeZonaPeligrosa(pedido);
        }

        if(pedido.getMascota()) {
            viaje=new DecoratorConMascota(viaje);
        }

        if(pedido.getEquipaje().equalsIgnoreCase( anotherString: "Baul")) {
            viaje=new DecoratorEquipajeBaul(viaje);
        }

        return viaje;
    }
}
```

Clases donde se implementa:

- UsuarioFactory
- ViajeFactory
- VehiculoFactory

Patrón MVC

Patrón MVC

Modelo:

- representa la lógica de negocio y el manejo de datos de la aplicación
- el modelo en esta aplicación es "Empresa"
- responsable de actualizar los estados de la aplicación basándose en las interacciones del usuario y las operaciones del sistema.

Patrón MVC

Vista

- Es la puerta de entrada al sistema para los usuarios
- Le permite ingresar datos a la aplicacion
- Le permite mostrar informacion a los distintos usuarios

Patrón MVC

Controlador

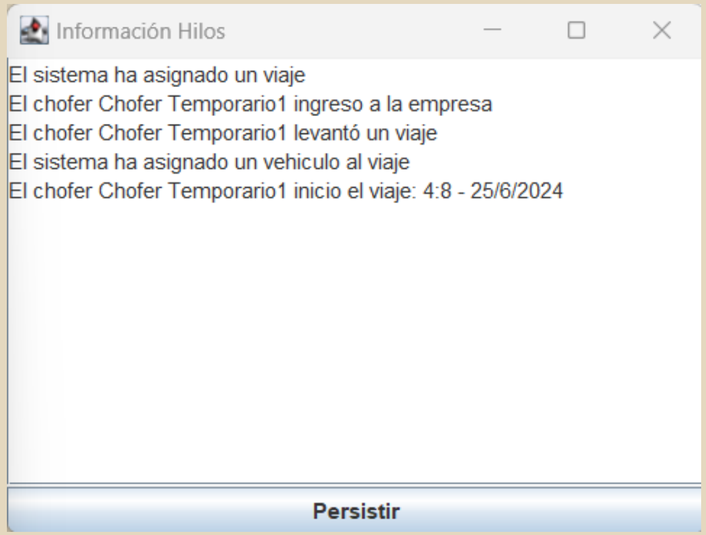
- Funciona de intermedierio entre la Vista y el Modelo
- Su funcion principal es analizar los eventos de la vista y saber que hacer con ellos
- Actualiza el modelo en base a los eventos de las vistas y actualiza las vistas en base a las actualizacion del modelo **(Observer/Observable)**

Patrón Observer / Observable

- Es un patron que define una relacion de uno a mas objetos
- Se utiliza para poder comunicar las actualizacion del Modelo a los controladores de las vistas

Persistencia

- Se crearon dos metodos, uno para guardar los datos de la aplicacion mediante la persitencia binaria



Hilos

- Se crearon 3 Hilos (HiloChofer, HiloCliente, HiloSistema) cada uno representa los 3 usuarios del sistema
- Se pueden crear varios Hilos choferes o hilos clientes dependiendo de los usuarios que la usen
- El Hilo Sistema se encuentra en la clase GestionPedido y es el encargado de elegir un chofer para cada viaje
- En el caso del uso mediante ventanas los hilos se crean con los controladores de cada chofer

Funcionamiento Simulacion

- Para ejecutar la Simulacion Basica solo hace falta ejecutar el objeto **Simulacion**
- Si queremos hacer uso del sistema nosotros mismo tendremos que ejecutar la clase **mainTesting** y interactuar con las distintas ventanas

Funcionamiento Simulación

Funcionamiento del codigo en Vivo

Muchas Gracias!!

Preguntas?